Санкт–Петербургский государственный университет

ГОРЕЛОВ Борис Романович

Выпускная квалификационная работа

Разработка системы удаленной компиляции

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2015 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Математическое и программное обеспечение вычислительных машин»

Научный руководитель:
доцент, кафедра технологии программирования
к.т.н. Блеканов Иван Станиславович

Рецензент:
доцент, кафедра компьютерных технологий
и систем, к.ф. - м.н. Смирнов Михаил Николаевич

Санкт-Петербург

2020 г.

Содержание

Введение 3

Постановка задачи 5

Глава 1. Обзор существующих решений 6

1.1. Введение 6

1.2. Visual Studio Code Remote SSH 6

1.3. Distcc 7

Глава 2. Архитектура системы 9

2.1. Введение 9

2.2. Сетевое взаимодействие 10

2.3. Архивация 11

2.4. Шифрование данных 11

2.5. Обслуживание сервера 13

2.6. Выборочная компиляция 14

2.7. Элементы автоматизации 14

Глава 3. Анализ полученной системы 16

3.1. Удобство использования 16

3.2. Скорость работы 16

3.3. Безопасность 16

3.4. Выявленные недостатки и пути улучшения 17

Заключение 18

Список источников 19

Приложение 1 21

Введение

Разработка программного обеспечения появилась вместе с первыми компьютерами всего несколько десятков лет назад. Тем не менее, этот род деятельности уже прошел невероятно длинный путь, технологии изменились до неузнаваемости. Возникают новые языки, меняются парадигмы программирования. Все это происходит для увеличения скорости и качества разработки, и, как следствие, снижения затрат.

Уже не один год процветает разработка программного обеспечения с открытым исходным кодом (open-source). Большое количество известных продуктов (например, браузер Firefox, операционная система Linux) разрабатываются таким образом. Любой желающий может их использовать на безвозмездной основе, а также участвовать в разработке, предлагать свои изменения. Участие в подобных проектах помогает набраться опыта и сделать в итоге полезный и качественный продукт, поэтому в будущем их популярность будет только возрастать.

С каждым годом сложность и объем программного обеспечения только растет. Зачастую количестве строк исходного кода в современных приложениях исчисляется миллионами. Компьютерные игры, фото/видео редакторы, браузеры - все это огромные проекты со сложной архитектурой. Большие размеры влекут за собой высокие требования к производительности систем, на которых ведется разработка.

Если говорить о компилируемых языках программирования, то как раз процесс компиляции является наиболее ресурсозатратным. Часто возникают ситуации, когда для тестирования нескольких изменений в коде с большим числом зависимых от него элементов приходится заново собирать весь проект. Такая ситуация становится неприемлемой, когда разработка ведется на условном ноутбуке со слабым процессором. Возникает естественная идея возложить обязанность компиляции кода на удаленный сервер. Это позволило бы экономить время сотрудников, сделать процесс разработки программного обеспечения более доступным.

В данной работе будут рассмотрены и проанализированы существующие подходы к решению задачи удаленной компиляции и предложен альтернативный подход, сочетающий в себе простоту использования и высокую скорость работы. Кроме того в разрабатываемой системе будут использованы методы обеспечения информационной безопасности.

Постановка задачи

Целью данной работы является разработка консольного клиент-серверного приложения, позволяющего клиенту скомпилировать файлы на серверном хосте.

В качестве операционной системы под которую будет вестись разработка выбран Linux. Данная операционная система обладает популярностью среди разработчиков по нескольким причинам. Благодаря тому, что Linux распространяется на бесплатной основе, он был портирован на большинство архитектур. Поэтому чаще всего именно эта операционная система функционирует на серверном оборудовании, бытовой и промышленной технике. Используя Linux на локальном компьютере разработчик получает окружение идентичное с тем, в котором будет работать его продукт. Кроме того, данная операционная система обладает удобной командной строкой, продуманной файловой системой и низкими требованиями к ресурсам.

Основными требованиями к разрабатываемому приложению являются:

- Простота и удобство использования
- Высокая скорость работы
- Обеспечение безопасности передачи данных

В задачи работы входит:

- Выявление преимуществ и недостатков существующих подходов к решению данной проблемы
- Разработка собственной системы, удовлетворяющей заявленным требованиям
- Анализ полученной системы

Глава 1. Обзор существующих решений

1.1 Введение

Технологии не стоят на месте и сейчас на рынке можно найти много готовых решений, позволяющих вести удаленную разработку и в том числе производить удаленную компиляцию. Условно решения данной задачи можно разделить на несколько классов. Один из них - это полноценные интегрированные среды разработки (IDE), предоставляющие широкие возможности, в том числе удаленную разработку (например NetBeans, CLion). К сожалению, часто для комфортной работы с большим проектом в таком приложении требуется мощный компьютер.

Другим классом решений являются утилиты, автоматизирующие сборку, с функционалом распределенных вычислений (например FASTBuild, IncrediBuild, Distcc). При наличии большого количества вычислительных мощностей данные утилиты многократно ускоряют процесс компиляции и сборки программного обеспечения, что является их главным плюсом. Среди недостатков можно отметить требование серьезной настройки программного окружения, узкий круг поддерживаемых языков (чаще всего только C/C++).

В данной главе будут рассмотрены и проанализированы несколько примеров, демонстрирующих разные подходы к решению.

1.2 Visual Studio Code Remote SSH

В качестве первого примера рассмотрим плагин Remote-SSH [1] для известного редактора кода VS Code. VS Code заслужил популярность благодаря своей гибкости и универсальности. Редактор работает на всех популярных операционных системах. Обширная библиотека расширений позволяет настроить редактор под конкретные задачи и приблизить инструмент по функционалу к уровню полноценной IDE. Плагин Remote-SSH позволяет использовать удаленный сервер для разработки, используя протокол

SSH [2]. Работа с использованием данного плагина строится по следующему принципу. Сперва производится начальная настройка SSH хоста. Далее на локальной машине открывается редактор и производится подключение к хосту. Таким образом открывается доступ к файловой системе и вычислительным мощностям удаленной машины.

К плюсам такого подхода можно отнести:

- Свободный доступ. Данное ПО распространяется бесплатно.
- Универсальность. Поддерживается большое количество языков программирования. Сервис совместим с большинством операционных систем.

Минусы:

- Нетривиальная конфигурация такой системы. Требуется установить совместимый SSH клиент (в случае работы под операционной системой Windows с этим могут возникнуть проблемы). Необходима первичная конфигурация хоста, а также переустановка на нем всех плагинов, использовавшихся локально.
- Результат работы сохраняется на удаленном сервере.

1.3 Distcc

Distcc [3] - программа для организации распределенной компиляции C/C++ кода. По своей сути данный проект является front-end(ом) для довольно популярного компилятора gcc [20]. По заявлению разработчиков, использование данной программы гарантирует одинаковый с локальной компиляцией результат, но дает значительный (в несколько раз) прирост в скорости. Время компиляции напрямую зависит от количества компьютеров, объединенных в сеть [4] (в данной материале показано, что в зависимости от бенчмарка, прирост скорости составляет приблизительно в 2

- 5 раз). Но даже обладая не большим вычислительным кластером, а одним удаленным компьютером, можно реализовать задуманную концепцию, производя компиляцию объемных файлов на нем.

Плюсы:

- Высокая производительность.
- Свободный доступ.

Минусы:

- Поддержка малого количества языков.
- Сложный процесс настройки.
- Низкое качество документации.

Глава 2. Архитектура системы

2.1 Введение

Из предыдущей главы становится ясно, что существует много подходов, позволяющих использовать удаленный хост для компиляции файлов. Зачастую это даже не основное предназначение программного продукта, а лишь побочный функционал. Как было отмечено, это влечет за собой некоторые недостатки: сложность конфигурации, низкая скорость работы. Описываемое приложение призвано решить эти проблемы.

В качестве основного языка разработки был выбран C. Данный язык зарекомендовал себя как лучшее средство в тех случаях, где важна производительность (операционные системы, драйверы). Большое количество уже реализованных качественных библиотек ускоряет процесс разработки и повышает его качество. Таким образом, данный выбор позволяет создать приложение, удовлетворяющее всем основным требованиям.

Также в ходе проектирования было решено остановиться на консольном приложении. Для упрощения работы и увеличения скорости взаимодействие с приложением сведено к минимуму, поэтому необходимость в графическом интерфейсе отсутствует. Кроме того, это позволяет ускорить разработку.

Система проектировалась таким образом, чтобы единственными необходимыми для запуска приложения клиентом параметрами были только IP адрес и номер порта сервера и название передаваемого файла. Для использования базового функционала клиентского приложения не требуется сложная конфигурация. Это позволяет реализовать возможность так называемой “работы из коробки”. С другой стороны, присутствует возможность использовать расширенный функционал. Пользователь может указать необходимые флаги компилятора, использовать возможность шифрования данных, о которой далее будет рассказано подробнее. С исходным кодом системы можно ознакомиться, перейдя по ссылке:

https://github.com/BorisGorelov/remote_compilation_system

2.2 Сетевое взаимодействие

Для всех сетевых взаимодействий был использован зарекомендовавший себя интерфейс сокетов Беркли [10]. Сокет позволяет управлять сетевым адресом, транспортом (то есть выбирать протокол транспортного уровня), а также предоставляет базовые операции ввода вывода (открытие/закрытие соединения, отправка/получение данных). В качестве протокола транспортного уровня при создании сокета было решено использовать TCP [7], так как в отличие от UDP он обеспечивает надежную доставку данных.

Работа сервера устроена следующим образом. Сперва создается сокет, связывается с определенным IP-адресом и портом. Далее вызывается метод listen(), который позволяет открыть данный порт на прослушивание, после чего сервер способен принимать входящие запросы на установку соединения. Для создания соединения с клиентом вызывается метод accept(). Важная деталь его работы заключается в том, что он создает новый сокет и связывает с клиентским сокетом именно его. Таким образом исходный серверный сокет используется только для приема входящих соединений, что дает возможность вести работу с несколькими клиентами.

Для того, чтобы отслеживать несколько клиентских сокетов, не блокируясь на них, использовался мультиплексор select. Он работает как обработчик прерываний, который активируется, как только на любой из сокетов приходят какие-либо данные. Если запрос пришел на исходный серверный сокет, он трактуется как запрос на новое соединение. Иначе происходит взаимодействие с уже существующим клиентом.

В серверной части важно отслеживать корректное выполнение всех этапов работы, поскольку некорректный пользовательский ввод или разрыв соединения могут привести к “зависанию”. Для отслеживания неполадок, произошедших на сервере, были проработаны коды ошибок сервера.

При возникновении нештатной ситуации, пользователю отправляется код, а также краткое информационное сообщение, что делает работу с приложением удобной.

Для каждого клиента на сервере создается отдельная папка, в которую сохраняются полученные файлы. После получения всех файлов вызывается компилятор и результат его работы записывается в отдельный лог с ошибками. Это сделано для удобства, потому что далеко не каждый раз программа успешно компилируется и пользователю важно знать, где именно возникла ошибка.

2.3 Архивация

Для удобства пользователя была добавлена возможность архивации данных. Если требуется обработать 1-2 файла, не составляет труда ввести их имена. В случае, если файлов больше, этот процесс может быть трудоемким, поэтому вместо файла пользователь может указать папку с файлами, содержащими исходный код. В этом случае с помощью архиватора tar [18] создается архив. Далее при помощи утилиты gzip [19] производится сжатие. В полученном виде информацию можно удобно и быстро передать на сервер для последующей работы.

2.4 Шифрование данных

В настоящее время при разработке любого ПО, работающего по сети, всегда особое внимание уделяют шифрованию данных. По сети ежедневно передается огромная масса личных данных (номера банковских карт, пароли, имена). Ведется переписка в социальных сетях, идет обмен данными по электронной почте. Без серьезного вмешательства криптографии в эти процессы нельзя рассчитывать, что передаваемая информация не будет доступна третьим лицам.

Учитывая вышесказанное, становится ясно, что передача данных в незащищенном виде может обернуться финансовыми убытками. Поэтому

при разработке описываемого приложения было принято решение добавить возможность шифрования данных.

Для этих целей использовалась широко известная криптографическая библиотека с открытым исходным кодом OpenSSL [12]. Как следует из названия, данная библиотека предоставляет доступ к реализации протоколов SSL/TLS [13]. TLS и его ныне устаревший предшественник SSL являются криптографическими протоколами, предназначенными для обеспечения безопасности связи в компьютерных сетях (по факту, большинство современных приложений использует TLS, но аббревиатура SSL прочно закрепилась в языке и до сих пор используется) . После установления соединения SSL гарантирует, что данные, передаваемые между сервером и клиентом, защищены и не повреждены. SSL используется многими банковскими приложениями, чтобы обеспечить безопасность при передаче конфиденциальных данных, таких как номер кредитной / дебетовой карты, имя пользователя и пароль.

SSL использует асимметричные алгоритмы шифрования для обеспечения безопасности передачи данных. В данных алгоритмах используются пары ключей (открытый и закрытый). Каждая сторона генерирует такую пару. Открытый ключ находится в свободном доступе. Закрытый же ключ известен только серверу или клиенту. В SSL данные, зашифрованные открытым ключом, могут быть расшифрованы только закрытым ключом, а данные, зашифрованные закрытым ключом, могут быть расшифрованы только открытым ключом.

Кратко рассмотрим процесс установки соединения. После того, как стороны договорились использовать TLS, согласование деталей соединения происходит с помощью процедуры рукопожатия. В начале клиент отправляет запрос на установление безопасного соединения и указывает в нем доступный ему набор шифров и хэш функций. Сервер выбирает шифр и хэш-функцию, отправляет подтверждение, а также свой цифровой сертификат [14], содержащий имя сервера, публичный ключ и название центра сертификации. Затем в случае, если клиент подтвердил действительность

сертификата, он шифрует случайное число публичным ключом сервера и отправляет его. Последним этапом сервер расшифровывает переданное случайное число своим приватным ключом и обе стороны с помощью данного числа генерируют сеансовый ключ для последующего шифрования и дешифрования данных во время сеанса. Таким образом обеспечивается защита данных даже в случае, если они были перехвачены третьей стороной.

С реализацией описанного выше процесса с помощью OpenSSL API можно ознакомиться в приложении 1. В нем представлены функции инициализации контекста, а также загрузки и проверки сертификата. Под инициализацией контекста понимается заполнение структуры SSL_CTX, являющейся фреймворком для установления соединений (в ней содержится информация об используемых протоколах, алгоритмах, сертификатах и т. д.), а также подгрузка необходимых сообщений об ошибках. В функции загрузки сертификатов происходит считывание данных из представленных данных и проверка на совместимость публичного и приватного ключа. В функции проверки сертификата происходит получение сертификата клиента (если он имеется) и печать имени клиента и центра сертификации.

2.5 Обслуживание сервера

Разработка системы велась с учетом того, что серверная часть должна работать непрерывно. Однако со временем возникнет ситуация, когда потребуется обновить систему. Чтобы данный процесс происходил удаленно и бесшовно было разработано несколько механизмов. Во-первых, пользователь может запросить рут-доступ для получения прав администратора. Для этого ему потребуется авторизоваться (все данные будут передаваться в защищенном виде). После успешной авторизации администратору высылается краткая статистика (количество пользователей в текущем сеансе, количество обработанных файлов), номер текущей версии и предоставляется возможность обновления системы. Для этого администратору потребуется загрузить обновленный код и в случае успешной компиляции произойдет замена старой программы на новую. Для этого был разработан

bash-скрипт, который удаляет старую версию и производит запуск новой.

2.6 Выборочная компиляция

Для обеспечения производительности был разработан механизм выборочной компиляции. Кратко его суть можно описать следующим образом: при повторной отправке компиляция производится только для измененных файлов. Дело в том, что часто возникает ситуация, когда изменился, например, один файл во всем проекте, и требуется протестировать работу этого изменения. В таких случаях целесообразно перекомпилировать только этот файл и заново собрать проект.

Рассмотрим принцип действия данного механизма. После получения файла сервер считает его хэш-сумму. Библиотека OpenSSL предоставляет большой выбор алгоритмов хэширования. В данном случае был выбран алгоритм md5 [11], так как он обеспечивает высокое качество (низкую вероятность коллизии) и сравнительно невысокую вычислительную сложность.

При повторном получении данного файла сначала сравнивается время модификации. Далее, если новая версия была модифицирована позднее старой, считается хэш-сумма нового файла, и только при несовпадении со старой производится повторная компиляция.

2.7 Элементы автоматизации

Для упрощения работы с разрабатываемым приложением были внедрены технологий автоматизации. Система должна сама “разобраться”, что пришло на вход и принять правильное решение о том, как его обработать. На первом этапе разработки было решено выбрать два языка для автоматической обработки: C и Java. Решение основано на анализе расширений полученных файлов.

Первым этапом проверяется наличие файла с инструкциями для утилит автоматизации сборки. Для сборки Java проектов поддерживается ути-

лита maven [15], для C - make [16]. В дальнейшем планируется расширение списка поддерживаемых утилит. Если среди переданных файлов встретился pom.xml (maven), либо makefile (make), вызывается соответствующая утилита.

Следующим этапом проверяются расширения. Если расширение переданных файлов .java, вызывается компилятор javac [17], преобразующий исходные файлы в байт-код, исполняемый JVM [17]. Если расширения переданных файлов .c и .h, используется компилятор gcc. По умолчанию подразумевается, что из переданных gcc файлов можно скомпоновать единый исполняемый файл.

Немаловажным этапом работы приложения является первичное развертывание. Некоторые из обсуждаемых выше утилит и библиотек не входят в стандартные дистрибутивы Linux, поэтому для их загрузки был разработан bash-скрипт. Помимо загрузки пакетов в его задачи входит создание цифровых сертификатов.

Глава 3. Анализ полученной системы

В данной главе будет произведен анализ полученной системы на предмет удовлетворения основным требованиям, а именно удобство использования, высокая скорость работы и безопасность. Также будут выявлены недостатки и пути возможного развития приложения.

3.1 Удобство использования

За счет того, что большинство процессов процессов протекают в автоматическом режиме, обеспечивается простота и удобство использования. Все необходимые библиотеки и утилиты загружаются и устанавливаются одной командой. Далее в базовом сценарии использования, когда клиенту нужно откомпилировать один файл, потребуется ввести только IP адрес и порт сервера и имя файла. Дальше система автоматически загрузит файл, выполнит необходимые команды и отправит результат своей работы обратно. В случае, если проект большой и сопровождается инструкциями для утилиты автоматизированной сборки, достаточно указать директорию, содержащую проект. Система создаст архив, произведет сжатие, отправит файлы на сервер и вернет результат сборки.

3.2 Скорость работы

Время, которое потребуется на выполнение всех этапов работы системы, несомненно, зависит от скорости интернет-соединения и вычислительной мощности сервера. Однако, даже несмотря на это, делегировав задачу компиляции у разработчика освобождается время для прочих дел.

3.3 Безопасность

В системе используется протокол TLS версии 1.2. Данная версия считается наиболее распространенной на данный момент и является достаточ-

но безопасной. Асимметричное шифрование обеспечивает конфиденциальность передачи данных (даже в случае перехвата на расшифровку сообщения уйдет неразумно большое время). Использование кодов аутентичности сообщений предотвращают возможность атак “человек посередине”.

3.4 Выявленные недостатки и пути улучшения

Разработанная система не лишена недостатков. В текущей версии поддерживается только два языка программирования. В последующих версиях планируется расширение количества поддерживаемых языков, а также утилит автоматизации сборки.

Благодаря тому, что основной функционал приложения написан на языке C, есть возможность реализовать кроссплатформенную версию приложения. Для этого потребуется доработать модуль архивации и добавить аналоги системных вызовов для соответствующих платформ.

Возможно в дальнейшем имеет смысл создание графического интерфейса приложения. Например, используя разработанную серверную часть можно реализовать систему для проведения небольшого чемпионата по олимпиадному программированию.

Заключение

В данной работе был проведен анализ существующих решений для организации процесса удаленной компиляции. Были выявлены основные недостатки, касающиеся трудоемкой конфигурации, узкой направленности решений.

Далее был произведен обзор компонентов и примененных подходов разработанной системы. Для обеспечения простоты и удобства использования были внедрены элементы автоматизации и возможности опциональной тонкой настройки. Выборочная компиляция, сжатие данных обеспечили высокую скорость работы приложения. Использование современных криптографических протоколов повысило надежность передачи данных.

Полученная система была проанализирована, выявлены сильные и слабые стороны. Представлены направления дальнейшей модернизации.

Поставленные задачи были выполнены, а цель работы достигнута.

Список источников

1. Visual Studio Code Remote-SSH extension -
   URL: https://code.visualstudio.com/docs/remote/ssh
2. Secure Shell cryptographic network protocol -
   URL: https://en.wikipedia.org/wiki/Secure_Shell
3. Distcc tool - URL: https://github.com/distcc
4. Benchmark results for distcc -
   URL: https://distcc.github.io/benchmark.html
5. W. Richard Stevens; Bill Fenner Andrew M. Rudoff (2003). Unix Network Programming, Volume 1: The Sockets Networking API (3rd Edition). Addison-Wesley Professional.
6. Advanced Programming in the UNIX environment, third edition, W. Richard Stevens and Stephen A. Rago, Addison-Wesley, 2013
7. Postel, Jon, "Transmission Control Protocol - DARPA Internet Program Protocol Specification RFC 793, DARPA, September 1981.
8. Linux Man pages - URL: https://www.kernel.org/doc/man-pages/
9. C language standard -
   URL: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf
10. Berkeley sockets -
    URL: https://en.wikipedia.org/wiki/Berkeley_sockets
11. MD5 message-digest algorithm -
    URL: https://en.wikipedia.org/wiki/MD5
12. OpenSSL Cryptography and SSL/TLS toolkit -
    URL: https://www.openssl.org

13. SSL/TLS cryptographic protocols -
URL: https://en.wikipedia.org/wiki/Transport_Layer_Security

14. Digital certificate -
URL: https://en.wikipedia.org/wiki/Public_key_certificate

15. Apache Maven - URL: http://maven.apache.org/

16. CNU Make - URL: https://www.gnu.org/software/make/

17. The Java programming language Compiler Group -
URL: http://openjdk.java.net/groups/compiler/

18. GNU Tar - URL: https://www.gnu.org/software/tar/

19. GNU Gzip - URL: https://www.gnu.org/software/gzip/

20. GNU GCC - URL: https://gcc.gnu.org/

Приложение 1

```
SSL_CTX*
InitServerContext(void)
{
    SSL_METHOD *method;
    SSL_CTX *context;
    SSL_load_error_strings();
    OpenSSL_add_all_algorithms();
    method = TLSv1_2_server_method();
    ctx = SSL_CTX_new(method);
    if ( context == NULL )
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    return context;
}
void
LoadCertificates(SSL_CTX* ctx, char* CertFile, char* KeyFile)
{
    if (SSL_CTX_use_certificate_file(ctx, CertFile, SSL_FILETYPE_PEM)
    <= 0)
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    if (SSL_CTX_use_PrivateKey_file(ctx, KeyFile, SSL_FILETYPE_PEM)
    <= 0)
    {
        ERR_print_errors_fp(stderr);
        abort();
```

```
    }
    if (SSL_CTX_check_private_key(ctx) == 0 )
    {
        fprintf(stderr, "Private key check failed\n");
        abort();
    }
}
void
ShowCertificates(SSL* ssl)
{
    X509 *certificate;
    char *line;
    cert = SSL_get_peer_certificate(ssl);
    if ( cert != NULL )
    {
        printf("Server certificates:\n");
        line = X509_NAME_oneline(X509_get_issuer_name(cert), 0, 0);
        printf("Issuer: %s\n", line);
        line = X509_NAME_oneline(X509_get_subject_name(cert), 0, 0);
        printf("Subject: %s\n", line);
        free(line);
        free(line);
        X509_free(cert);
    }
    else
        printf("No certificates.\n");
}
```